RIACS

Research Institute for Advanced Computer Science
NASA Ames Research Center

IN-60
43064
P25

Development of a Sensor Coordinated Kinematic Model for Neural Network Controller Training

Charles C. Jorgensen
Research Institute for Advanced Computer Science
NASA Ames Research Center

August 1990

RIACS Technical Report 90.28

NASA Cooperative Agreement Number NCC 2-387

(NASA-CR-188864) DEVELOPMENT OF A SENSOR COORDINATED KINEMATIC MODEL FOR NEURAL NETWORK CONTROLLER TRAINING (Research Inst. for Advanced Computer Science) 25 pCSCL 09B

N91-32799

Unclas

G3/60 0043064

Development of a Sensor Coordinated Kinematic Model for Neural Network Controller Training

Charles C. Jorgensen

Research Institute for Advanced Computer Science
NASA Ames Research Center[1]

RIACS Technical Report 90.28

NASA Cooperative Agreement Number NCC 2-387

1. Work reported herein was supported by Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA) with contributions from the Information Sciences Division of Ames Research Center and the Information Systems Division of Langley Research Center.

Abstract

This paper presents a robotic benchmark problem useful for evaluating alternative neural network controllers (see companion report Jorgensen 1990). Specifically, it derives two camera models and the kinematic equations of a multiple degree of freedom manipulator whose end effector is under observation. The mappings developed include forward and inverse translations from binocular images to 3-D target positions and the inverse kinematics of mapping point positions into manipulator commands in joint space. Implementation is detailed for a three degree of freedom manipulator with one revolute joint at the base and two prismatic joints on the arms. The example is restricted to operate within a unit cube with arm links of.6 and .4 units respectively. The development is presented in the context of more complex simulations and a logical path for extension of the benchmark to higher degree of freedom manipulators is presented. Source code implementing the equations in MATLAB is included in a supplementary appendix[2].

2. I would like to express my appreciation to the members of the Sparse Distributed Memory Group for the many stimulating discussions during this work. In particular I would like to acknowledge Dr. Doug Danforth for his suggestion to simplify the learning rate proportion term in the self organizing maps and his analytical insights about alternative neighborhood designs. Finally, I would like to thank Dr. Mike Raugh for creating a research environment making this work possible.

Foreward

An issue of growing importance in teleoperations is the integration of multiple sensor input with control manipulators. In addition to calibration and image interpretation issues, there often arise questions of how the necessary control relationships are determined. This is particularly true as the complexity of the manipulator and the operational environment increases. In certain situations (e.g. high degrees of freedom, dynamic environments, or non-linearities) traditional methods available to formally specify controllers may not be computationally efficient.

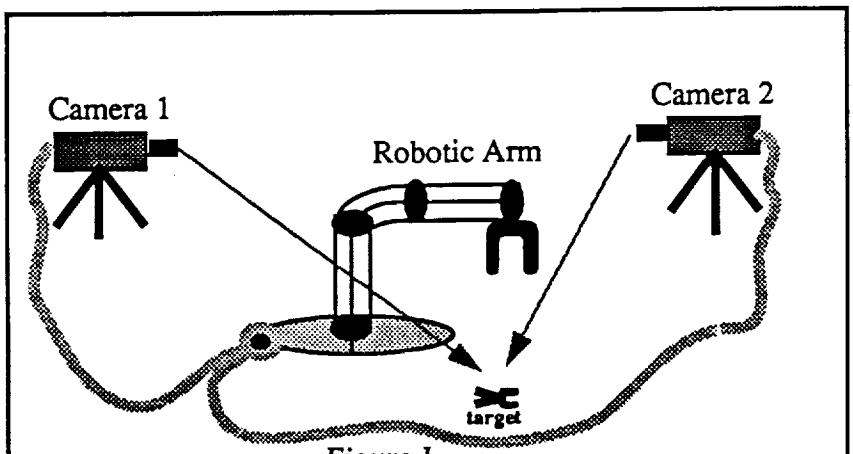


Figure 1
Coordinated Movement

Figure 1 shows a drawing of a typical situation which might occur in space based assembly or autonomous exploration. Two cameras are located at variable positions in a work space. The task involves having the cameras focus on a target and record coordinates for input to a controller which translates the values received into joint angles. Two mappings are required to complete a control sequence. First, the relationships between the two camera images must be learned so a pair of 2-D images correspond to a single 3-D point. Second, inverse kinematics determined by the particular arm parameters must be learned to map the 3-D point coordinates to a set of end effector values in joint space. Because of potentially complex relationships between the number of degrees of freedom and the possible joint angles, this mapping does not always have a unique inverse, that is there may be many different joint positions and sequences which can arrive at the same point in space.

In the present paper a composite model of two cameras and a robotic arm is developed which is intended to be used to produce training data for a neural network controller operating in such an environment. Encoding the model requires a number of assumptions. These are detailed below.

Camera Model

Given a vector of target coordinates, we require a function Θsuch that $\Theta(x, y, z) \Rightarrow (x_1, y_1, x_2, y_2)$ where the subscripts refer to cameras 1 and 2 respectively. Figure 2 illustrates a typical relationship for two viewpoints relative to a single focal plane

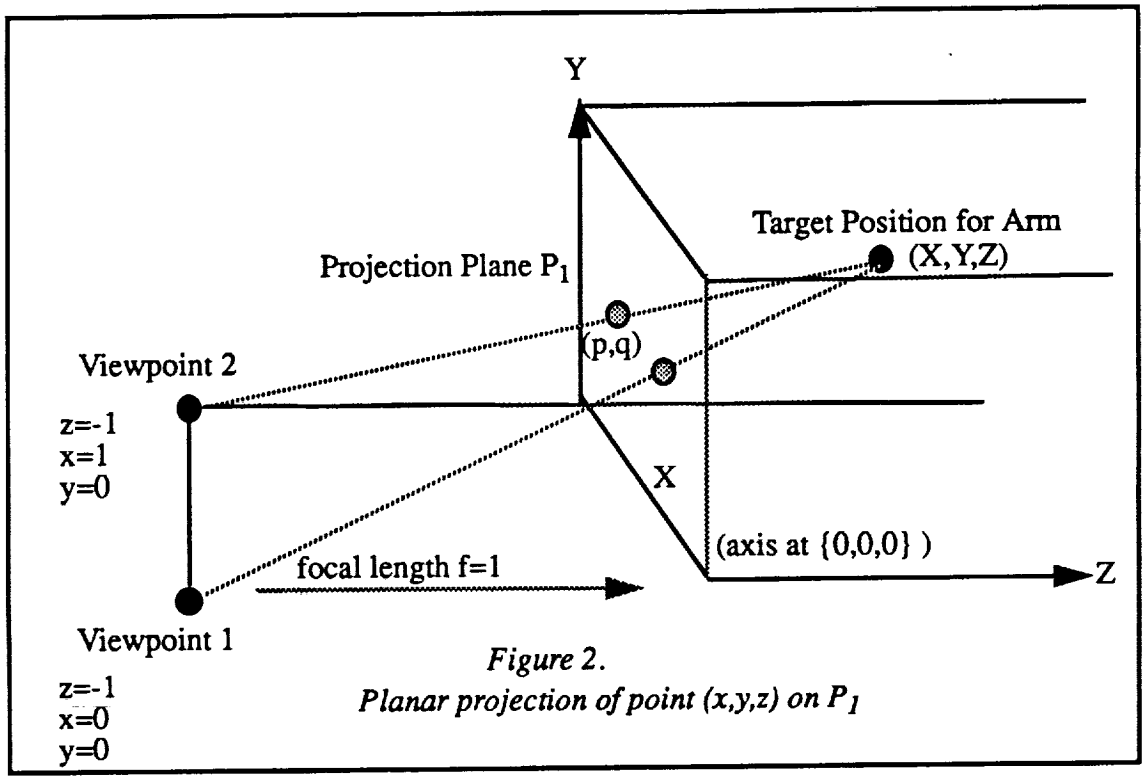


Figure 2.
Planar projection of point (x,y,z) on P_1

It can be seen that if we wish to relate multiple cameras to a common target we may need to take into account the angle of the camera relative to the axis of the scene, the field size onto which the image is projected, the distance between the centers of the cameras and the focal point locations.

Fortunately, effective methods have been developed for handling such transformations (i.e. rotation, skew, scale, and translation) using homogeneous coordinate systems[3]. For producing neural controller training data, we simplify this figure by allowing each of the two viewpoints to lie on the same plane, i.e. the Z-X axis of a target object's coordinate system[4]. For example, each camera could be placed so its projection plane P is parallel to theY axis in figure 2. If we make a further restriction of unit focal length, then the focal plane coordinates (p,q) are related to the location of the target in the manipulator space (x,y,z) as below where a,b, and c are functions of the focal length and define the camera model [5], its position, and its orientation. The extra dimension is a homogeneous coordinate.

3. Ballard, D. and Brown, C, "Computer Vision" Prentice Hall, 1982.

4. Longuet-Higgins, H.C., "A Computer Algorithm for Reconstructing a Scene from Two Projections" In Readings in Computer Vision, Morgan Kaufmann, 1987.

$$\begin{bmatrix} p \\ q \\ 1 \end{bmatrix} = \frac{1}{(c_1x + c_2y + c_3z + c_4)} \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

If we further let the focal plane be the axis of one of the dimensions and place the cameras a distance 1 unit from each other starting at $x = 0$, then the relationship between the focal plane coordinates of projection one (p_1,q_1) and the target point (x,y,z) in figure 2 reduces to:

$$\{p_1, q_1\} = \left\{ \frac{x}{(1+z)}, \frac{y}{(1+z)} \right\}$$

Similarly for camera 2 and point (p_2,q_2):

$$\{p_2, q_2\} = \left\{ \frac{1-x}{1+z}, \frac{y}{1+z} \right\}$$

These equations can be used to develop planer coordinate projections for arbitrary target points. It is necessary to also know their inverse transformation, i.e. the set of equations which must be learned by the neural network to determine the original x,y,z target coordinates from the camera projections only.These can be shown to be:

$$z = \left(\frac{1}{p_2 + p_1} \right) - 1 \qquad x = \frac{p_1}{p_1 + p_2} \qquad y = \frac{q_1}{p_1 + p_2}$$

To test only integration of vision with kinematics, the complexity of the equations can be reduced considerably. Because the current problem treats objects as positioning points to be reached by the arm, perspective distortions on the camera plane will not come into consideration. If desired they can be introduced into the model by post multiplying the coordinates by the following equation where w is the additional dimension required for a homogeneous coordinate system and f is the focal length from the image plane.

$$(x', y', z', w') = (x, y, x, w) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5. A discussion of the more complex perspective transforms can be found in "Differential and Integral Projective Invariants in Imagery", Barrett, E.B., Payton, P.M., and Brill, M.H., SPSE 43rd Annual Conference, May 20-25, 1990, Rochester Institute of Technology.

Arm Models

Besides specification of the projections from a target point, it is necessary to generate a model of a robot arm. For the distributed neural control problem, the goal was a three degree of freedom arm moving within a unit volume. This objective was approached beginning with a two and three DOF planar manipulators, and finally a three DOF non planar manipulator.

There are a number of decisions when generating higher degree of freedom models. To clarify why the present model was chosen, some background is helpful. A robot arm is just a combination of links and joints formed together in a chain with one end fixed and one free. Each joint is driven by an actuator. The free end, also called an end effector is moved along a path by sequentially activating the joints. Thus it is necessary to know the displacement of a joint at each point in time with respect to a fixed reference axis called the base frame. A path for the end effector is defined in terms of the movements relative to this frame. One of the current popular approaches for such arm modelling is called the computed torque method.[6] In this approach, the dynamics of a rigid robot manipulator are described by a vector differential equation,

$$M(\theta)\,\ddot{\theta} + C(\theta, \dot{\theta}) + G(\theta) + D(\theta, \dot{\theta}, t) = \tau$$

where:

θ is an nx1 vector of joint coordinates

$M(\theta)$ is an n x n inertia matrix

$C(\theta, \dot{\theta})$ is the nx1 vector of Coriolis and centrifugal forces

$G(\theta)$ is the nx1 vector of gravitational forces

$D(\theta, \dot{\theta}, t)$ is the nx1 vector of disturbances including friction and

τ is the nx1 vector of generalized torques applied by actuators at the manipulator joints

In practice τ is usually factored into two parts. The first is composed of the inertia matrix and the second is composed of nonlinearities in the manipulator dynamics. To simplify neural network training simulations, compliance is often not considered, non linear terms are minimized, and the fact that the inertia matrix is symmetric and positive definite used to guarantee that it is invertible.[7] In practice, a real world arm will not behave exactly as predicted by a model, hence adjustment of parameters is required.

---

6. Salam, F., "On the Design and Control of Robot Manipulators", Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics, vol 1, 1986, pg.195-199.

7. Paul, R.P., "Robot Manipulators: Mathematics, Programming, and Control," Addison-Wesley, Reading, MA, 1986.

One implication of such adjustments is that for space based systems, adaptive neural controllers might be useful to compensate for unanticipated changes. It would be particularly advantageous if the controller could be both adaptive and trainable in situations where frequent task reconfigurations or teleoperations may be required. Consequently consideration of how to generate an error term to correct learning also becomes necessary. Using the above framework, if we let the total required torque tau be represented by two parts - a linear and a non linear:

$$\tau = \alpha\tau' + \beta$$

where $\alpha = \hat{M}(\theta)$ and $\beta = \hat{C}(\theta, \dot{\theta}) + \hat{G}(\theta) + \hat{D}(\theta, \dot{\theta}, t)$ (the hat signifies the best available model of the corresponding terms in the dynamic equation) and if the right hand side of the equation is substituted for τ and kinematic equation rearranged, we get:

$$\hat{M}(\theta)\ (\ddot{\theta} - \tau') = \Delta M\ddot{\theta} + \Delta C + \Delta G + \Delta D(\tau)$$

where:

$$\Delta M = \hat{M}(\theta) - M(\theta)$$

$$\Delta C = \hat{C}(\theta, \dot{\theta}) - C(\theta, \dot{\theta})$$

$$\Delta D(t)) = \hat{D}(\theta, \dot{\theta}, t) - D(\theta, \dot{\theta}, t)$$

$$\Delta G = \hat{G}(\theta) - G(\theta)$$

Because the inertia terms are invertible, the mismatch between observed values for $\theta, \dot{\theta}, \ddot{\theta}$ and estimates from their best available models reduces to

$$\ddot{\theta} - \tau' = \hat{M}(\theta)^{-1}\ (\Delta M\ddot{\theta} + \Delta C + \Delta G + \Delta D(t))$$

So, to achieve a desired path and velocity, $(\theta_d \dot{\theta}_d)$ the torques should be selected such that they are a proportion of the difference between the observed and estimated values for angle, velocity and acceleration of the joints, specifically:

$$\tau' = \ddot{\theta}_d + K_v(\dot{\theta}_d - \dot{\theta}) + K_a(\theta_d - \theta)$$

If we define error E as the difference between the desired and actual angles for the joints $E = (\theta - \theta_d)$ we arrive at an error term suitable for weight adjustment in a neural network controller.

$$Error = \ddot{E} + K_v\dot{E} + K_aE$$

Previous research (Jorgensen and Schley 1989) indicated the vulnerability of certain classes of neural network algorithms to nonlinear properties of the control functions (in

the case of aircraft these were manifest during landing under wind shear). One of the conclusions of this study was that function fitting networks would fail at singularity points where an underlying control law had abrupt transitions. Such situations occur for arm kinematics as well because limits on joint movements, velocities, and accelerations create discontinuities i.e. zeros within the characteristic polynomial. Thus to control an arm, a neural network must provide either an underlying structure capable of implementing the complete characteristics of the underlying kinematic equation (a PD controller) or for initial tests we must simplify the arm. Since the ability of neural nets to successfully capture arbitrary non linear functions from observation is still very much an unresolved issue, it appeared best to approach the model of the arm gradually using increasingly realistic models. Figure 3 shows a simplified initial arm configuration. We began with a two DOF planar manipulator with joint angles θ_1, θ_2 and links $link_1$ and $link_2$. Assuming

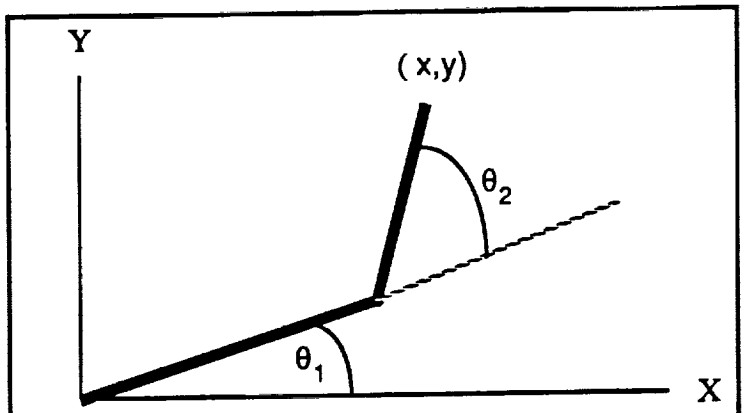


Figure 3.
Two DOF Planar Manipulator

non compliant links and omitting non linear terms, the forward end effector positioning kinematics become:

$$x = link_1 \cos(\theta_1) + link_2 \cos(\theta_2 + \theta_1)$$

$$y = link_2 \sin(\theta_1 + \theta_2) + link_1 \sin(\theta_1)$$

Because of the degree of freedom and the manipulator geometry, closed forms can also be derived for the inverse kinematics of this manipulator. They are:

$$\theta_2 = \cos^{-1}\left(\frac{(x^2 + y^2 - link^2_1 - link^2_2)}{2 link_1 link_2}\right)$$

$$\theta_1 = \tan^{-1}\left(\frac{y}{x}\right) - \tan^{-1}\left(\frac{link_2 \sin(\theta_2)}{link_1 + link_2 \cos(\theta_2)}\right)$$

To train this manipulator for the problem defined above we would normally have to learn first a camera coordinate model and second an arm control law for a set of training paths in the work space.

The camera coordination problem does not apply to this simpler manipulator because a point coordinate is the same as the camera coordinate i.e. an orthographic projection on the coordinate axis. To generate the control training sets, a convenient method is to teach the manipulator to follow circles within the 2-D space. These are derived from: $x = r\cos(\alpha)$ and $y = \sin(\alpha)$. A circular set of x,y coordinates is then generated by letting α vary in discrete steps over a circle radius r.

The above equations used separately provide sufficient information to train neural network alternatives for the two types of components of the test problem. In the first, each net is evaluated to see if it can learn 3-D stereopsis for the two cameras. In the second, the networks would learn the manipulator control problem.

Increasing complexity can be introduced into the exercise in a number of ways. First the camera models can be made more realistic by allowing arbitrary location, lens distortion, number of perspectives, and calibration.

The kinematic problem can also be made more realistic by increasing the number of degrees of freedom, introducing the non linear components of the torque calculations, or compliance. For example, if we increase the complexity to a 3 DOF planar manipulator (Figure 4) we can solve the forward kinematics using:

$$y = link_1 \cos(\theta_1) - link_2 \cos(\theta_1 + \theta_2) - link_3 \cos(\theta_1 + \theta_2 + \theta_3)$$

$$x = link_1 \sin(\theta_1) - link_2 \sin(\theta_1 + \theta_2) - link_3 \sin(\theta_1 + \theta_2 + \theta_3)$$

However, because we now have a three degree of freedom manipulator operating in a 2-D space, a redundant degree of freedom is introduced and hence a non invertible relationship. Thus even the use of a simple planar arm permits non-trivial tests of neural control concepts.[8] As soon as the model of the arm moves from planar to volumetric, changes in the individual joint coordinate systems must be taken into account.[9]

8. Guez, A. and Ziauddin, A. "Improving the Solutions of the Inverse Kinematic Problem in Robotics Using Neural Networks", Journal of Neural Network Computing, Vol 1, No. 4, 1990.

9. Hollerbach, J.M. "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity", IEEE SMC Vol 10,No. 11, 1980.

This can be accomplished by fixing a coordinate frame for each joint and propagating a change in orientation and axis through the remaining links to the end effector.Two specifications are required, one for the link and one for the joints.

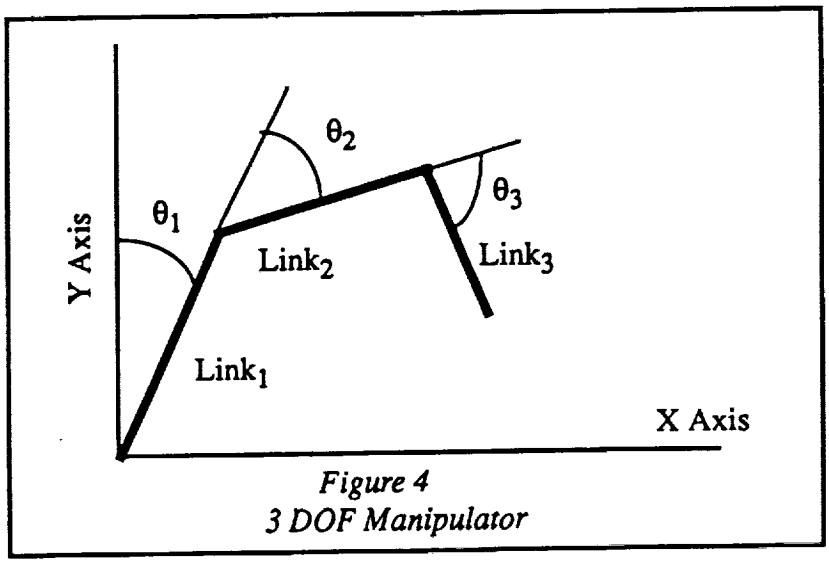


Figure 4
3 DOF Manipulator

Because certain hardware configurations may require specific cross sections for strength and bends for joint connections, the links need to be characterized by two dimensions: a common normal distance a_n between the joints (called length) and the angle α_n between the axes in a plane perpendicular to a_n also called the twist[10]. Similarly, joints may be characterized by the distance d_n between two consecutive joints and θ_n the angle between the two links. Two kinds of joints may need to be considered: revolute and prismatic. In the case of the revolute joints their potential to the modify the axis creates a more complex coordinate geometry than the prismatic. Depending upon the definitions of the axis, in general the final end effector position can be found in terms of a series of rotations and translations such as the following:

1. Rotate about the z axis an angle θ_n

2. Translate along the z axis a distance d_n

3. Translate along the now rotated x axis a length a_n

4. Rotate about the translated x axis a twist angle α_n

These operations can be expressed in terms of a product of four homogeneous transformations relating the coordinate frame of link n to the coordinate frame of link n-1 through a matrix of rotations often called the A matrix, defined for revolute and prismatic joints respectively as:

10. Paul, R. and Shimano, B., "Kinematic Control Equations for Simple Manipulators," IEEE Trans. On SMC, VOL. SMC-11, NO.6, June 1981.

$$A_{revolute,\,n} = \begin{bmatrix} \cos(\theta) & -\sin(\theta)\cos(\alpha) & \sin(\theta)\sin(\alpha) & a\cos(\theta) \\ \sin(\theta) & \cos(\theta)\cos(\alpha) & -\cos(\theta)\sin(\alpha) & a\sin(\theta) \\ 0 & \sin(\alpha) & \cos(\alpha) & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_{prismatic,\,n} = \begin{bmatrix} \cos(\theta) & -\sin(\theta)\cos(\alpha) & \sin(\theta)\sin(\alpha) & 0 \\ \sin(\theta) & \cos(\theta)\cos(\alpha) & -\cos(\theta)\sin(\alpha) & a\sin(\theta) \\ 0 & \sin(\alpha) & \cos(\alpha) & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If we let the coordinate frame for the end effector be represented by U_n, the description of its frame of reference is given by:

$$U_n = A_1 \times A_2 \times A_3 \ldots \times A_n$$

And if the end effector is related to an arbitrary reference base R such as the space station work platform by a transformation Z and it is assumed to be grasping a tool described in terms of reference coordinates T, we can describe the end of the tool with respect to the reference system by:

$$R = Z \times U_n \times T$$

3-D Arm

For neural network purposes, a simpler arm can be generated by ignoring some of the dimensions of the full models. In particular, complexity can be dramatically reduced by omitting link twist, permitting a revolute joint only at the base, and two prismatic joints, one connecting the base to the first link of a planar manipulator, and one connecting the first link to the second. The result is a simple manipulator such as figure 5.

The forward and inverse kinematics of this manipulator can be derived as follows. The forward relationships can be solved trigonometrically through three intermediate terms based on theta one (the angle the arm is rotated on the x,y plane), theta 2 (the angle the first link makes with the x,y plane), and theta 3 (the angle link two makes with link one). First we define length d of a path between the center of the base point and link 2 and the angle theta tilde that this link makes with the x,y plane as:

$$d = \sqrt{(link_1)^2 + (link_2)^2 - 2 link_1 link_2 \cos(\theta_3)}$$

$$\tilde{\theta} = \theta_2 - \text{acos}\left(\frac{(d^2 + (link_1)^2 - (link_2)^2)}{2 d link_1}\right)$$

We can then solve for the length of the radial projection r this segment makes on the x,y plane using:

$$r = d\cos(\tilde{\theta})$$

from which it follows that:

$$z = d\sin(\tilde{\theta}) \qquad y = r\cos(\theta_1) \qquad x = r\sin(\theta_1)$$

In a similar fashion, we can derive a closed form inverse kinematic for this

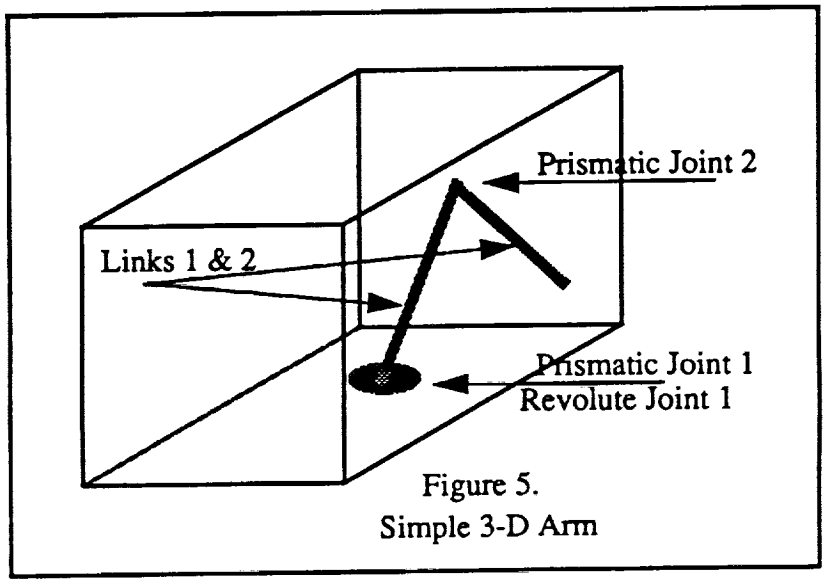


Figure 5.
Simple 3-D Arm

manipulator by defining d as:

$$d = \sqrt{y_2 + x_2 + z_2}$$

$$\theta_3 = \text{acos}\left(\frac{((link_1)^2 + (link_2)^2 - d^2)}{2link_1 link_2}\right)$$

$$\theta_2 = \text{asin}\left(\frac{z}{d}\right) + \text{acos}\left(\frac{(d^2) + (link_1)^2 - (link_2)^2}{2dlink_1}\right)$$

$$\theta_1 = \text{atan}\left(\frac{x}{y}\right)$$

Discussion

Based on the above benchmark specifications, the required relationships are specified to generate data for training alternative neural network controllers. We initially assumed two cameras separated by a distance 2d from each other and parallel to the same plane of observation. Training points were generated by setting arm parameters at .6 for line one, .4 for link 2, and joint limits. The set of reachable points is created by stepping the arm through each possible joint position and using the forward kinematic model to calculated the x,y,z coordinate that was reached by the end effector. These points are in turn presented to the camera models which generated the corresponding sets of planar coordinates.

Once a set of reachable points for a particular model had been determined, training and test sets could be picked at random. If the neural controller can learn stereopsis, the first half of the learning capabilities would be demonstrated. Once there was an established target point in 3D space, the second requirement is to train arm controller in the inverse kinematic relationships. This is accomplished by using the solved coordinates in the camera task and generating a teaching set of corresponding joint angles from the inverse kinematic equations. The set of correct joint angles can then be used as an error reference against which the output of the neural controller is adjusted during training.

Thus, learning for the neural network has of two component parts shown in Figure 6. As can been seen, the problem can be solved using either two separate networks or a single net taking camera input directly and outputting joint coordinates. Depending upon the nature of the operational environment, there may be advantages to either approach. For example, if it is known that part of the processes are linear, faster convergence methods can be implemented during training. If on the other hand, some of the non linear processes can be anticipated, preprocessing transforms may reduce the complexity of the problem and a single network might be most efficient. Another situation would be when adaptive

control is required and not all parts of the system have to adjust at the same rates. In these cases multiple neural controllers rather than a single all purpose controller would be appropriate. Such a situation is considered in the companion to this report.

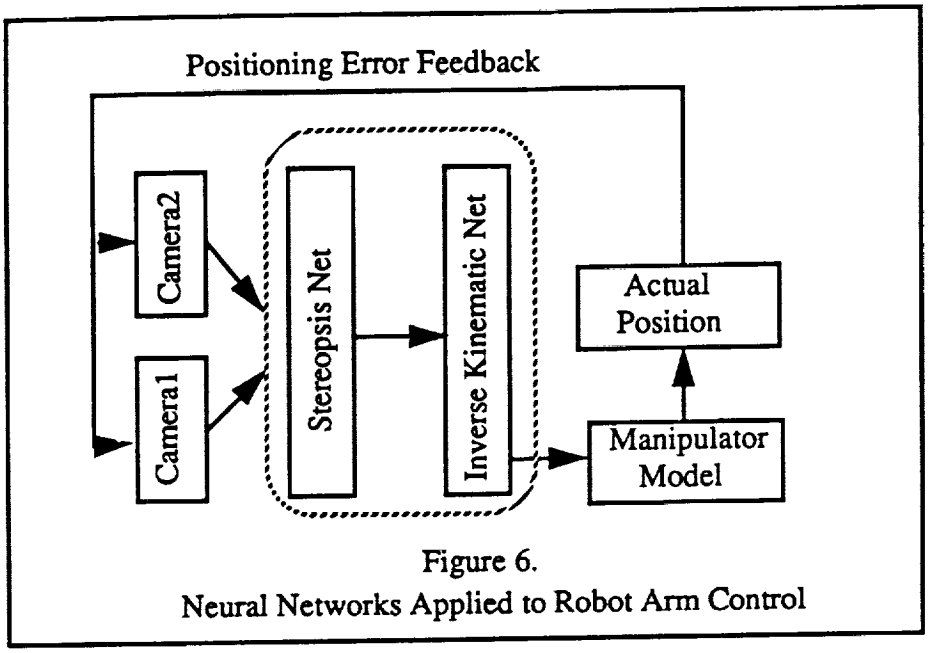


Figure 6.
Neural Networks Applied to Robot Arm Control

Once a type of trainable network has been selected (suing function fitting or distributed representations) and a learning paradigm chosen (supervised or unsupervised) slightly different training strategies may be required. For example, in the case of a distributed network such as an SDM[11] or CMAC[12], one difficulty is the correct number of training samples required to represent the control equation smoothly. In the case of another type of network called a self organizing map[13], a central question is whether the training sample represents the underlying probability distribution of the process being sampled. In both cases, the evaluation process involves training the network through presentations of part of the point samples and evaluating the network's performance on the remaining points.

What is most useful about the above benchmark is that it provides test conditions which can be incrementally increased in complexity. Complexity can be manipulated along dimensions of the number of variables, the degree of linearity or non linearity present in the functions to be approximated, and the fidelity of the hardware models. It thus represents a natural paradigm for moving from theoretical investigations of the potentials of alternative network controller concepts to field applications.

11. Kanerva, P. "Sparse Distributed Memory," MIT Press, Cambridge, MA, 1988.
12. Albus, J.S., "A Theory of Cerebellar Functions," Mathematical Biosciences, 10(1/2):25-61, 1971.
13. Kohonen, T. "Self Organization and Associative Memory," Spring Series in Information Sciences, Heidelberg, 1984.

The following appendix contains MATLAB code which was generated to implement the above benchmark. Also included is a plot of the planar projections of a set of points reachable by a particular set of parameters for the 3-D arm model. Because this arm was treated as though it were in the corner of a room reaching outward toward points in a unit cube, the x,y projection forms a quarter circle arc. The base joint was stepped through rotational angles in three degree increments, so the projection of points forms a set of radiating straight lines. The projections on the other axis correspond the density of reachable points for that particular arm configuration. A change in joint lengths, joint angle limits, and sampling rates would produce a different set of projections. The projections are included for illustrative reasons because they are a useful visual aid when trying to observe how well a particular neural controller is learning the distribution of training points. A discussion of the results of applying of this particular benchmark to two new types of neural network controllers can be found in "Distributed Memory Approaches for Robotic Neural Controllers," (Jorgensen, RIACS Technical Report, 1990).

Appendix

(MATLAB Routines Used to Model Camera and Arm Dynamics)

```
function output=build3dpts(arm1,arm2,stepsize)
% Routine to generate a sample of x,y,z points in a volume of size
% sizex, sizey, sizez, in step sizes of stepsize. This routine
% is used to build a series of data points fed to the inverse
% camera and robot arm models to create a training set for the neural
% controller. Note stepsize is the coarseness of the grid in radians

% The routine begins by generating points that are legal arm
% angles for each joint. This may require some though to assure the
% end effector remains in the positive quadrant of the space
% Next using these angles, the legitimate values x,y, and z can
% take on are decided by inverse kinematic calculations for the arm
% finally the camera model is given these points to calculate the
% lens projections that would have occurred if they were in fact
% being observed by the cameras

count=1

% set arm sizes
% e.g. arm1=.6;
% e,g  arm2=.4;

% set camera focal length, and lens separations
focallength=1;
separation=.3;

% assure angle range on matrix dimensions, I am setting them so
% the revolute joint goes from 0 to 90 degrees
% the first prismatic joint can go from 0 to 90-arc sin (arm1)degree
% and the second prismatic joint can go from 0 to 180 degrees
% stepsize is the number of radians per joint move and in effect sets the
% fineness of the data sampling grid for the learning function
% Set joint ranges to avoid quadrants that produce imaginary roots
% begin loops
for revangle=.001:stepsize:pi/2
for prismatic1=.001:stepsize:.001:stepsize:((pi/2)-(asin(arm2/arm1)))
for prismatic2=.001:stepsize:(pi-(pi/180))
theta1=revangle;
theta2=prismatic1;
theta3=prismatic2;
[x,y,z]=robot3df(arm1,arm2,revangle,prismatic1,prismatic2);
%  call inverse camera routine to calculate image plane projections
[x1 y1 x2 y2]=camera(focallength,separation,x,y,z);
output(count,1:10)=[x y z x1 y1 x2 y2 theta1 theta2 theta3];
count=count+1;
end
count
end
end
```

```
function tset=buildtset(data,numpoints)
% Routine to build subsets of arbitrary size from a given training set
% used to test RBSDM
a=length(data);
for i=1:numpoints
tset(i,:)=data((ceil(rand*a)),:);
end
```

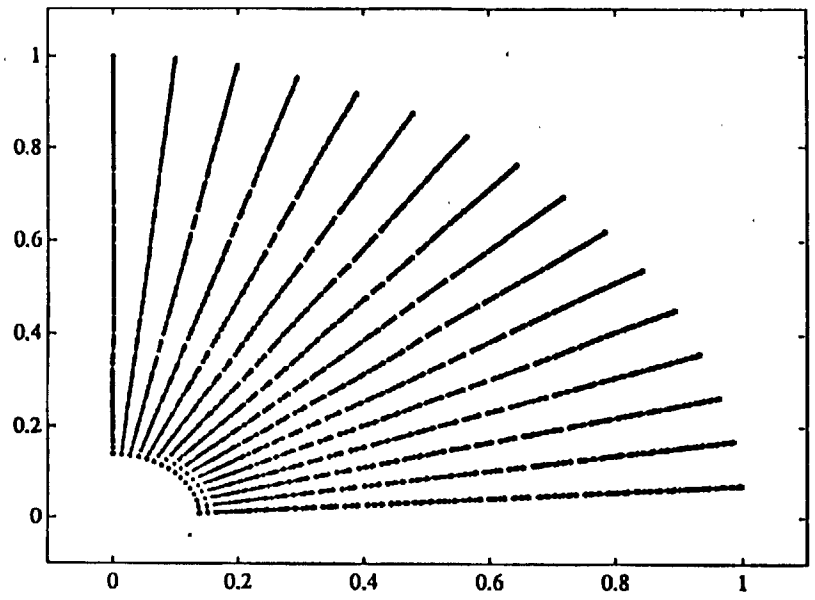
```
function [x1,y1,x2,y2]=camera(f,d,x,y,z)
%
% Function to produce 2D projections by two cameras on their image planes
% from 3D target point (see Jorgensen 1990 for derivation rational and limits)
% (inverse projections only)
% f is camera focal lenght  d is one half the distance between cameras axes
% x,y,z are the spatial coordinates of the arm target

x1=(f*(x-d))/(f-z);
y1=(f*y)/(f-z);
x2=((x+d)*f)/(f-z);
y2=(f*y)/(f-z);
```

```
function [x,y,z]=camerafwd(f,d,x1,y1,x2,y2)

% Routine to calculate forward x,y,z coordinates from two sets of
% 2D images. Assumes both cameras in same plane axis on same side of
% target at focal length f and distance 2d apart

z=f-((2*d*f)/(x2-x1))
x=(x1*(f-z)/f)+d
y=(y2*(f-z))/f
```

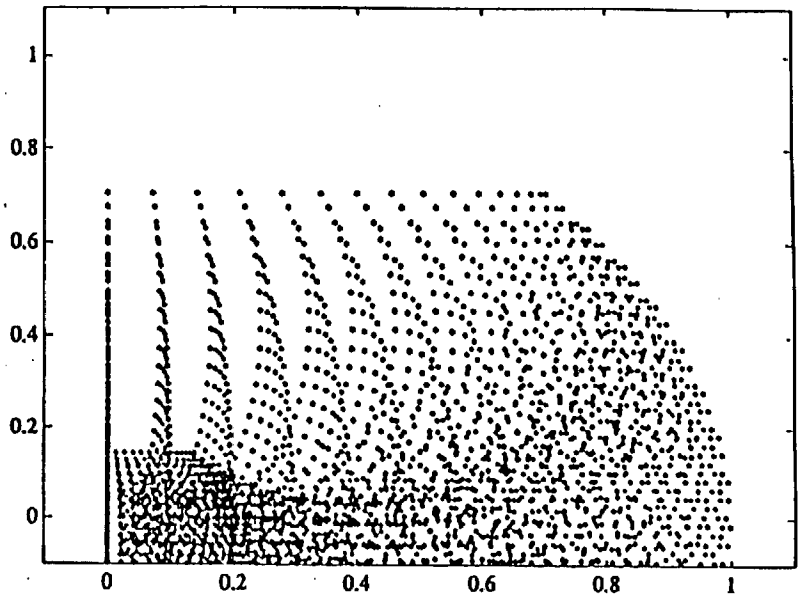
```
function [x,y,z]=robot3df(l1,l2,theta1,theta2,theta3)
% routine to calculate forward kinematics for 3dof manipulator with
% base revolute joint and 2 prismatic joints one at base and the other
% between links two and three

% define length of path between base and end effector
d=sqrt((l1^2)+(l2^2)-(2*l1*l2*cos(theta3)));
% define the angle this segment makes with the y,x plane
tstar=theta2 - acos(((d^2)+(l1^2)-(l2^2))/(2*d*l1));
% find length of projection of this segment on the y,x plane
r=d*cos(tstar);
%calculate the 3dof height using this angle and d
z=d*sin(tstar);
% calculate the y, and x values using theta one and r
y=r*cos(theta1);
x=r*sin(theta1);
```

```
function [theta1,theta2,theta3]=robot3dinv(l1,l2,x,y,z)

% Function to calculate inverse kinematics for a 3 dof manipulator
% in 3D space with a revolute base joint and two prismatic joints
% and two links
% x,y,z are the end effector coordinates in 3D space
% l1 is the first link from the base point, l2 is the second link
% theta1 is the revolute angle of the base
% theta2 is the angle of the first prismatic joint relative to the
% 3D x,y plane and theta 3 is the second prismatic joint angle relative
% to link1

d=sqrt((y^2)+(x^2)+(z^2));
theta3=acos(((l1^2)+(l2^2)-(d^2))/(2*l1*l2));
theta2=asin(z/d)+acos(((d^2)+(l1^2)-(l2^2))/(2*d*l1));
theta1=atan(x/y);
```

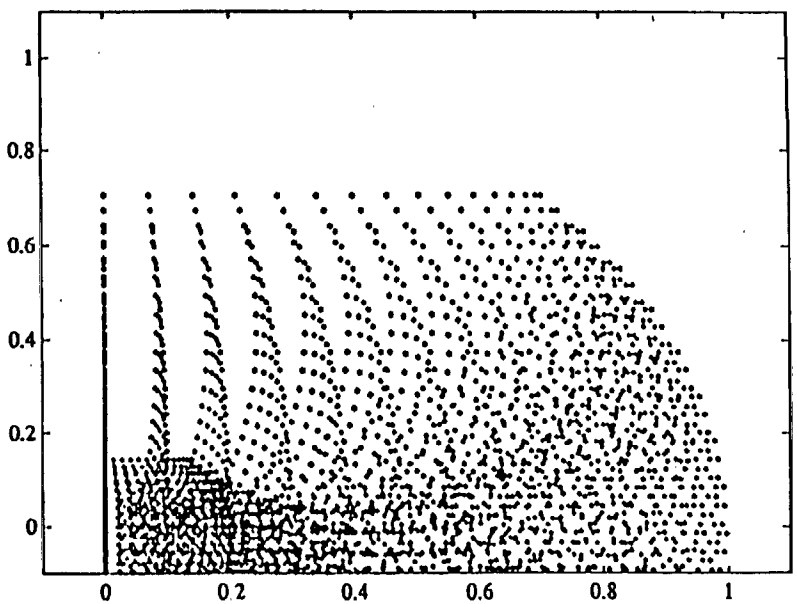
Axis Projection of Original Data Sets



XY Axis Projection



XZ Axis Projection



YZ Axis Projection